Human Activity Recognition

Can different human activities be distinguished using data automatically collected by sensors attached to the actor’s body? We explore this question with a Weight Lifting Exercise Dataset described here: http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har)

The data concerns dumbbell bicep curls done in one of five different manners (one correct, five incorrect) by 6 young subjects. If poor exercise technique can be automatically detected and diagnosed, efficiencies in training for exercises could be achieved. More information on published work with the data is avaiable here:

Velloso, E.; Bulling, A.; Gellersen, H.; Ugulino, W.; Fuks, H. Qualitative Activity Recognition of Weight Lifting Exercises. Proceedings of 4th International Conference in Cooperation with SIGCHI (Augmented Human ’13) . Stuttgart, Germany: ACM SIGCHI, 2013.

Getting the data

The data is available for downloading from the internet.

```
if (!file.exists("pmltraining.csv")){
        urlTrain <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
        download.file(url=urlTrain, destfile="./pmltraining.csv")
}
if (!file.exists("pmltesting.csv")){
        urlTest <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
        download.file(url=urlTest, destfile="./pmltesting.csv")
}

pmltraining <- read.csv("./pmltraining.csv")
pmltesting <- read.csv("./pmltesting.csv")
```

```
dim(pmltraining)
```

```
## [1] 19622   160
```

```
dim(pmltesting)
```

```
## [1]  20 160
```

It’s a large data set, 19622 cases and 160 variables. There are 20 cases provided as “unkowns” to used as test cases.

The first 159 variables are potential predictors. Variable 160 is the response variable, “classe”, for the training set. Variable 160 is the variable, “problem_id”, for the testing set, giving the problem number for the 20 case testing set, keyed to the 20 problems to be submitted to Coursera.

```
class(pmltraining[,160])
```

```
## [1] "factor"
```

```
table(pmltraining[,160])
```

```
## 
##    A    B    C    D    E 
## 5580 3797 3422 3216 3607
```

There are 5 response levels, A, B, C, D, and E. Given values for the first 159 variables, our task is to predict the response A, B, C, D, and E.

Cleaning the data

There are plenty of NAs in the data. Let's quantify that observation, checking for each variable the fraction of NAs.

```
naCount = integer()
naFraction = numeric()
numcoltraining <-  ncol(pmltraining)
numrowtraining <-  nrow(pmltraining)
for(n in 1:numcoltraining) {
        naCount[n] <- sum(is.na(pmltraining[,n]))
        naFraction[n] <- naCount[n]/numrowtraining
}
checknas <- data.frame(names(pmltraining), naFraction)
checknas
```

```
##           names.pmltraining. naFraction
## 1                          X     0.0000
## 2                  user_name     0.0000
## 3       raw_timestamp_part_1     0.0000
## 4       raw_timestamp_part_2     0.0000
## 5             cvtd_timestamp     0.0000
## 6                 new_window     0.0000
## 7                 num_window     0.0000
## 8                  roll_belt     0.0000
## 9                 pitch_belt     0.0000
## 10                  yaw_belt     0.0000
## 11          total_accel_belt     0.0000
## 12        kurtosis_roll_belt     0.0000
## 13       kurtosis_picth_belt     0.0000
## 14         kurtosis_yaw_belt     0.0000
```

```
## 15        skewness_roll_belt     0.0000
## 16      skewness_roll_belt.1     0.0000
## 17         skewness_yaw_belt     0.0000
## 18              max_roll_belt     0.9793
## 19             max_picth_belt     0.9793
## 20               max_yaw_belt     0.0000
## 21              min_roll_belt     0.9793
## 22             min_pitch_belt     0.9793
## 23               min_yaw_belt     0.0000
## 24        amplitude_roll_belt     0.9793
## 25      amplitude_pitch_belt     0.9793
## 26        amplitude_yaw_belt     0.0000
## 27      var_total_accel_belt     0.9793
## 28              avg_roll_belt     0.9793
## 29           stddev_roll_belt     0.9793
## 30              var_roll_belt     0.9793
## 31             avg_pitch_belt     0.9793
## 32          stddev_pitch_belt     0.9793
## 33             var_pitch_belt     0.9793
## 34               avg_yaw_belt     0.9793
## 35            stddev_yaw_belt     0.9793
## 36               var_yaw_belt     0.9793
## 37               gyros_belt_x     0.0000
## 38               gyros_belt_y     0.0000
## 39               gyros_belt_z     0.0000
## 40               accel_belt_x     0.0000
## 41               accel_belt_y     0.0000
## 42               accel_belt_z     0.0000
## 43              magnet_belt_x     0.0000
## 44              magnet_belt_y     0.0000
## 45              magnet_belt_z     0.0000
## 46                   roll_arm     0.0000
## 47                  pitch_arm     0.0000
## 48                    yaw_arm     0.0000
## 49            total_accel_arm     0.0000
## 50              var_accel_arm     0.9793
## 51               avg_roll_arm     0.9793
## 52            stddev_roll_arm     0.9793
## 53               var_roll_arm     0.9793
## 54              avg_pitch_arm     0.9793
## 55           stddev_pitch_arm     0.9793
## 56              var_pitch_arm     0.9793
## 57                avg_yaw_arm     0.9793
## 58             stddev_yaw_arm     0.9793
## 59                var_yaw_arm     0.9793
## 60                gyros_arm_x     0.0000
## 61                gyros_arm_y     0.0000
```

```
## 62                gyros_arm_z     0.0000
## 63                accel_arm_x     0.0000
## 64                accel_arm_y     0.0000
## 65                accel_arm_z     0.0000
## 66               magnet_arm_x     0.0000
## 67               magnet_arm_y     0.0000
## 68               magnet_arm_z     0.0000
## 69          kurtosis_roll_arm     0.0000
## 70         kurtosis_picth_arm     0.0000
## 71           kurtosis_yaw_arm     0.0000
## 72          skewness_roll_arm     0.0000
## 73         skewness_pitch_arm     0.0000
## 74           skewness_yaw_arm     0.0000
## 75               max_roll_arm     0.9793
## 76              max_picth_arm     0.9793
## 77                max_yaw_arm     0.9793
## 78               min_roll_arm     0.9793
## 79              min_pitch_arm     0.9793
## 80                min_yaw_arm     0.9793
## 81         amplitude_roll_arm     0.9793
## 82        amplitude_pitch_arm     0.9793
## 83          amplitude_yaw_arm     0.9793
## 84              roll_dumbbell     0.0000
## 85             pitch_dumbbell     0.0000
## 86               yaw_dumbbell     0.0000
## 87     kurtosis_roll_dumbbell     0.0000
## 88    kurtosis_picth_dumbbell     0.0000
## 89      kurtosis_yaw_dumbbell     0.0000
## 90     skewness_roll_dumbbell     0.0000
## 91    skewness_pitch_dumbbell     0.0000
## 92      skewness_yaw_dumbbell     0.0000
## 93          max_roll_dumbbell     0.9793
## 94         max_picth_dumbbell     0.9793
## 95           max_yaw_dumbbell     0.0000
## 96          min_roll_dumbbell     0.9793
## 97         min_pitch_dumbbell     0.9793
## 98           min_yaw_dumbbell     0.0000
## 99    amplitude_roll_dumbbell     0.9793
## 100 amplitude_pitch_dumbbell     0.9793
## 101   amplitude_yaw_dumbbell     0.0000
## 102     total_accel_dumbbell     0.0000
## 103       var_accel_dumbbell     0.9793
## 104        avg_roll_dumbbell     0.9793
## 105     stddev_roll_dumbbell     0.9793
## 106        var_roll_dumbbell     0.9793
## 107       avg_pitch_dumbbell     0.9793
## 108    stddev_pitch_dumbbell     0.9793
```

```
## 109       var_pitch_dumbbell     0.9793
## 110         avg_yaw_dumbbell     0.9793
## 111      stddev_yaw_dumbbell     0.9793
## 112         var_yaw_dumbbell     0.9793
## 113         gyros_dumbbell_x     0.0000
## 114         gyros_dumbbell_y     0.0000
## 115         gyros_dumbbell_z     0.0000
## 116         accel_dumbbell_x     0.0000
## 117         accel_dumbbell_y     0.0000
## 118         accel_dumbbell_z     0.0000
## 119        magnet_dumbbell_x     0.0000
## 120        magnet_dumbbell_y     0.0000
## 121        magnet_dumbbell_z     0.0000
## 122             roll_forearm     0.0000
## 123            pitch_forearm     0.0000
## 124              yaw_forearm     0.0000
## 125   kurtosis_roll_forearm     0.0000
## 126  kurtosis_picth_forearm     0.0000
## 127    kurtosis_yaw_forearm     0.0000
## 128   skewness_roll_forearm     0.0000
## 129  skewness_pitch_forearm     0.0000
## 130    skewness_yaw_forearm     0.0000
## 131        max_roll_forearm     0.9793
## 132       max_picth_forearm     0.9793
## 133         max_yaw_forearm     0.0000
## 134        min_roll_forearm     0.9793
## 135       min_pitch_forearm     0.9793
## 136         min_yaw_forearm     0.0000
## 137  amplitude_roll_forearm     0.9793
## 138 amplitude_pitch_forearm     0.9793
## 139   amplitude_yaw_forearm     0.0000
## 140     total_accel_forearm     0.0000
## 141       var_accel_forearm     0.9793
## 142        avg_roll_forearm     0.9793
## 143     stddev_roll_forearm     0.9793
## 144        var_roll_forearm     0.9793
## 145       avg_pitch_forearm     0.9793
## 146    stddev_pitch_forearm     0.9793
## 147       var_pitch_forearm     0.9793
## 148         avg_yaw_forearm     0.9793
## 149      stddev_yaw_forearm     0.9793
## 150         var_yaw_forearm     0.9793
## 151         gyros_forearm_x     0.0000
## 152         gyros_forearm_y     0.0000
## 153         gyros_forearm_z     0.0000
## 154         accel_forearm_x     0.0000
## 155         accel_forearm_y     0.0000
```

```
## 156          accel_forearm_z     0.0000
## 157         magnet_forearm_x     0.0000
## 158         magnet_forearm_y     0.0000
## 159         magnet_forearm_z     0.0000
## 160                   classe     0.0000
```

Thus it turns out that the variables divide neatly into two sets, those with no missing data and those with over 97 percent NAs. We begin to build the cleaner *training* and *testing* sets we will actually use by selecting just the variables without missing data.

```
goodvarsL <-  naFraction < 0.01
sum(goodvarsL)
```

```
## [1] 93
```

```
max(naCount[goodvarsL])
```

```
## [1] 0
```

```
training <- pmltraining[, names(pmltraining)[goodvarsL] ]
testing <- pmltesting[, names(pmltesting)[goodvarsL] ]
dim(training)
```

```
## [1] 19622    93
```

```
dim(testing)
```

```
## [1] 20 93
```

We have reduced the number of variables from 160 to 93.

The variable X is a unique identifier for the cases, so it is not useful for prediction. We drop it.

```
training <- training[ , !names(training)=="X"]
testing <-  testing[ , !names(testing)=="X"]
dim(training)
```

```
## [1] 19622    92
```

```
dim(testing)
```

```
## [1] 20 92
```

We will build our predictor with a random forest. We plan to use the caret package, which calls on the randomForest, which at present does not accept categorical variable with 32 or more levels. Since there are some in the training data set, we remove them.

```
numcol <- ncol(training)
varkeep <- rep(TRUE, numcol)
for (n in 1:(numcol-1) ) {
        temp <- training[ ,n]
        if ( class(temp) == "factor" )   {
                if (nlevels(temp) >= 32) {
                        varkeep[n] <- FALSE
                }
        }
}
sum(varkeep)
```

```
## [1] 68
```

So we only keep 67 predictor variables and the 1 response variable.

```
train1 <- training[,varkeep]
test <- testing[,varkeep]
dim(train1)
```

```
## [1] 19622    68
```

But now we note that 9 of the variables in the 20 case test set are missing all data. We remove those variables from the data set.

```
badTestVars <- c("kurtosis_yaw_belt", "skewness_yaw_belt", "amplitude_yaw_belt", "kurtosis_yaw_
dumbbell", "skewness_yaw_dumbbell", "amplitude_yaw_dumbbell", "kurtosis_yaw_forearm", "skewness
_yaw_forearm", "amplitude_yaw_forearm" )
train1 <- train1[,!(names(train1) %in% badTestVars )]
dim(train1)
```

```
## [1] 19622    59
```

```
test <- test[,!(names(test) %in% badTestVars )]
dim(test)
```

```
## [1] 20 59
```

So in the end we only keep 58 predictor variables and the 1 response variable.

Separating off a cross-validation set for checking out-of-sample accuracy.

Because it takes so long to run the full data set, we run it on only part of the data. We take a random selection of 10000 of the 19622 observations

```
set.seed(12345)
nsample <- 10000
samples <- sample(nrow(train1), size = nsample)
train1 <- train1[samples,]
```

We partition our training data set train1 into two subsets: train2 contains 90% of the cases and will be used to train the predictor. crossval2 contains 10% of the cases and will be used to make an out of sample estimate of the accuracy.

```
long <- nrow(train1)
trainrows <- sample(long, size = round(0.9*long))
crossvrows <- c(1:long)[is.na(pmatch(x=c(1:long), table = trainrows))]
head(trainrows)
```

```
## [1] 2444 6894 8695 9810 5691 1643
```

```
head(crossvrows)
```

```
## [1] 45 66 76 83 86 94
```

```
train2 <- train1[trainrows,]
crossval2 <- train1[crossvrows,]
dim(train2)
```

```
## [1] 9000   59
```

```
dim(crossval2)
```

```
## [1] 1000   59
```

Building and evaluating the predictor

We use the caret package to build a random forest predictor. It takes about 5 hours to run on our reduced data set, so we save it as an RDS file for future use, enabling us to deactivate the train command command during final editing, instead reading in the modFit1 file the train command produced the first time through.

```r
library(caret)
```

```
## Loading required package: lattice
## Loading required package: ggplot2
```

```r
modFit1 <- train(classe~ ., data=train2, method="rf", prox=TRUE)
modFit1
saveRDS(modFit1, "model10000alt.RDS")
```

```r
modFit1 = readRDS("model10000alt.RDS")
modFit1
```

```
## Random Forest
##
## 9000 samples
##   58 predictors
##    5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
##
## Summary of sample sizes: 9000, 9000, 9000, 9000, 9000, 9000, ...
##
## Resampling results across tuning parameters:
##
##   mtry  Accuracy  Kappa  Accuracy SD  Kappa SD
##   2     1         1      0.002        0.003
##   40    1         1      0.001        0.002
##   80    1         1      0.002        0.002
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 41.
```

Now for a prediction on the cross validation set. We are able to compare the predictions with the true values on 1000 cases that are not part of the training set.

```r
predictCrossVal <- predict(modFit1, crossval2)
```

```
## Loading required package: randomForest
## randomForest 4.6-7
## Type rfNews() to see new features/changes/bug fixes.
```

```
rightOrWrong <- predictCrossVal == crossval2[, "classe"]
table(rightOrWrong)
```

```
## rightOrWrong
## FALSE  TRUE
##     1   999
```

```
accuracy <- sum(rightOrWrong)/length(rightOrWrong)
accuracy
```

```
## [1] 0.999
```

The predictor was correct in 999 out of 1000 cases in the cross-validation set, an accuracy of 99.9%.

Predicting the unknown test set

And finally, the predictions for the twenty problems with unknown solution.

```
predictVars1 <- predict(modFit1, test)
predictVars1
```

```
##  [1] B A B A A E D B A A B C B A E E A B B B
## Levels: A B C D E
```